Programmer = démontrer?
La correspondance de Curry-Howard aujourd'hui

Premier cours

Les chemins d'une découverte: la correspondance de Curry-Howard, 1930–1970

Xavier Leroy

Collège de France

2018-11-21

COLLÈGE
DE FRANCE
1530

Le chemin vers Curry-Howard

- Le lambda-calcul
  Initialement une logique (terme = proposition !), devenu le proto-langage de programmation fonctionnelle.
- La logique intuitionniste
  Naissance de l'idée du ≪terme de preuve≫ qui matérialise une démonstration.
- Les types simples
  De la théorie des types de Russell au lambda-calcul simplement typé.
- Curry, 1957
  Propositions = types ; prouvabilité = habitation
  (dans le cas particulier des combinateurs).
- Howard, 1969–1980
  Propositions = types ; démonstrations = termes ;
  élimination des coupures = réductions.

I

Le lambda-calcul

La lambda-notation

Vers 1930, Alonzo Church introduit la notation $\lambda x.\, e$
pour ≪la fonction qui au paramètre x associe l'expression e≫.

Aujourd'hui on noterait $x \mapsto e$. Mais cette notation, popularisée par Bourbaki, est plus récente.

Ces notations $\lambda x.\, e$ ou $x \mapsto e$ sont bien utiles pour clarifier le discours mathématique sur les fonctions…

≪ la fonction x^2 ≫	$x \mapsto x^2$ ou $\lambda x.\, x^2$
≪ la fonction $x^2 + y^2$ ≫	$(x, y) \mapsto x^2 + y^2$ ou $\lambda x.\lambda y.\, x^2 + y^2$
≪ la famille de fonctions cx^2 ≫	$c \mapsto (x \mapsto cx^2)$ ou $\lambda c.\lambda x.\, cx^2$

Calculer avec la lambda-notation

Deux règles de calcul essentielles :

α-conversion : renommage d'une variable liée

$$\lambda x.\, M =_\alpha \lambda y.\, M\{x \leftarrow y\} \qquad \text{si } y \text{ non libre dans } M$$

β-conversion : substitution du paramètre formel par l'argument effectif dans le corps d'une fonction.

$$(\lambda x.\, M)(N) =_\beta M\{x \leftarrow N\}$$

Là ou en mathématiques usuelles on écrirait

Soit la fonction $f(x) = x + 1$. *On a* $f(2) = 3$.

la lambda notation décompose ce calcul en étapes :

$$f(2) \;=\; (\lambda x.\, x + 1)\,(2) \;=_\beta\; 2 + 1 \;=\; 3$$

Pourquoi lambda ?

Dana Scott raconte avoir écrit à Church

> *Dear professor Church : why “lambda” ?*

et reçu pour réponse

> *Eeny, meeny, miny, moe.*

Rosser dit que Church a simplifié progressivement la notation $\hat{x}e$ utilisée dans *Principia Mathematica* de Russell et Whitehead :

$$\hat{x}e \quad \to \quad \wedge xe \quad \to \quad \lambda xe \quad \to \quad \lambda x[e] \quad \to \quad \lambda x.e$$

Un lambda pour les lier tous

La variable x dans $\lambda x.e$ est dite «liée» et peut être renommée sans changer le sens. Il en va de même de nombreuses notations mathématiques assez disparates :

$$\{x \mid P(x)\} \qquad \forall x, P(x) \qquad \exists x, P(x)$$

$$\bigcup_{n \in N} A(n) \qquad \prod_{i=1}^{n} i \qquad \sum_{n=0}^{\infty} \frac{1}{n^2}$$

$$\lim_{n \to \infty} u_n \qquad \int_a^b f(x)dx$$

Un lambda pour les lier tous

Church propose de traiter toutes ces notations comme des opérateurs prenant une lambda-abstraction en argument :

$$\begin{aligned}
\forall x, P(x) &\equiv \forall(\lambda x.\, P(x)) \quad \text{(ou juste } \forall P\text{)} \\
\exists x, P(x) &\equiv \exists(\lambda x.\, P(x)) \\
\sum_{n=0}^{\infty} \frac{1}{n^2} &\equiv \mathtt{sum}(0, \infty, \lambda n. \frac{1}{n^2}) \\
\int_a^b f(x)dx &\equiv \mathtt{integr}(a, b, \lambda x. f(x))
\end{aligned}$$

Ainsi il n'y a plus qu'une seule construction qui lie des variables : la lambda-abstraction.

C'est la naissance de la syntaxe abstraite d'ordre supérieur (*Higher-Order Abstract Syntax*).

Le premier lambda-calcul de Church (1932, 1933)

Une logique, proposée comme alternative à la théorie des ensembles, où

- prédicats et propositions sont écrits en lambda-notation ;
- les prédicats peuvent prendre des prédicats en arguments (logique d'ordre supérieur) ;
- les ensembles A peuvent donc être représentés comme leurs prédicats d'appartenance $x \mapsto x \in A$.

(On est donc dans le schéma ≪propositions = termes≫.)

Ensembles = prédicats

Les opérations ensemblistes s'expriment en termes de prédicats et d'application de prédicats :

$$
\begin{array}{lrcl}
\text{Appartenance :} & x \in A & \equiv & A\,x \quad \text{(application)} \\
\text{Compréhension :} & \{x \mid P(x)\} & \equiv & P \\
\text{Ensemble vide :} & \emptyset & \equiv & \lambda x.\, 0 \\
\text{Singleton :} & \{x\} & \equiv & \lambda y.\, x = y \\
\text{Union binaire :} & A \cup B & \equiv & \lambda x.\, A\,x \vee B\,x \\
\text{Union générale :} & \bigcup_{x \in A} B(x) & \equiv & \lambda y.\, \exists x, A\,x \wedge B\,x\,y
\end{array}
$$

Le paradoxe de Russell s'exprime aussi :

$\lambda x.\, \neg(x\,x)$	l'ensemble de tous les x tq $x \notin x$ …
$(\lambda x.\, \neg(x\,x))\ (\lambda x.\, \neg(x\,x))$	… appartient-il à lui-même ?

Le premier lambda-calcul de Church

A. Church, Postulates for the Foundation of Logic, *Annals of Math.* 33(2), 1932.

A. Church, Postulates for the Foundation of Logic, *Annals of Math.* 34(4), 1933.

Termes : $M, N ::= x$ variable

$\mid \lambda x.\, M$ abstraction de fonction

$\mid M\ N$ application de fonction

$\mid \Pi(M, N)$ $\forall x, M\ x \Rightarrow N\ x$

$\mid \Sigma(M)$ $\exists x, M\ x$

$\mid \&(M, N)$ M et N

$\mid \tilde{}(M)$ non M

$\mid \iota(M)$ un x tel que $M\ x$

$\mid A(M, N)$???

\+ 5 règles de déduction, dont α-conversion et β-conversion

\+ 37 axiomes (36 dans la version 2).

Incohérence logique

Le paradoxe de Russell s'exprime : si $U = (\lambda x.\ \neg(x\ x))\ (\lambda x.\ \neg(x\ x))$, on a $U =_\beta \neg U$. Cela n'introduit pas d'incohérence car la logique de Church est une logique intuitionniste minimale.

L'incohérence logique est prouvée par :

- Kleene et Rosser, 1935.
  Un codage compliqué du paradoxe de Richard.
- Curry, 1942.
  Soit A une proposition. On considère $D = \lambda x.\ x\ x \Rightarrow A$ et $U = D\ D$.
  Alors $U \Leftrightarrow (U \Rightarrow A)$, ce qui implique A.

C'est la fin du lambda-calcul (non typé) comme logique.
Mais d'autres applications apparaissent…

Le lambda-calcul pur

Vers 1935, Church et ses étudiants Kleene et Rosser étudient le lambda-calcul «pur», débarassé de ses connecteurs logiques :

$$M, N ::= x \mid \lambda x.\, M \mid M\ N$$

α-équivalence et β-réduction (= conversion orientée) :

$$\lambda x.M =_\alpha \lambda y.M\{x \leftarrow y\} \qquad (\lambda x.M)\ N \to_\beta M\{x \leftarrow N\}$$

(y non libre dans M)

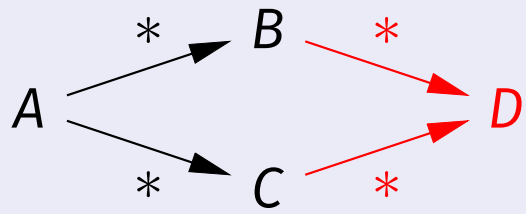
Une notion d'exécution apparaît : itérer la β-réduction jusqu'à une «forme normale» N irréductible.

$$M \to_\beta M_1 \to_\beta M_2 \to_\beta \cdots \to_\beta N \not\to_\beta$$

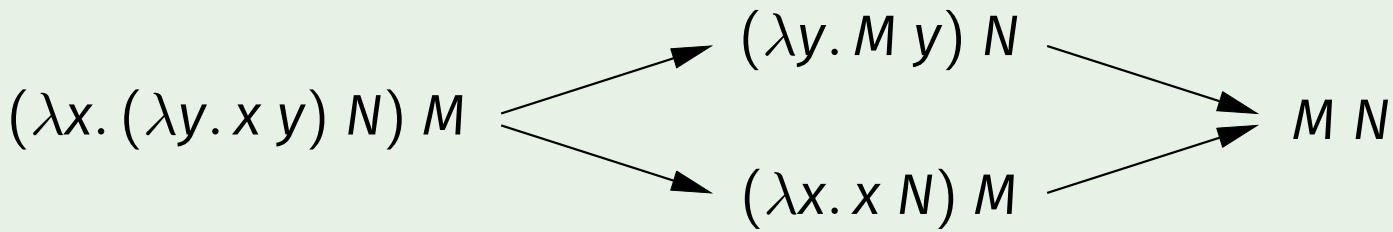
Bonnes propriétés des réductions

Théorème (Church et Rosser, 1935)

La β-réduction est confluente.



Exemple



Corollaire

Si elle existe, la forme normale d'un terme est unique.

Lambda-calcul et calculabilité

Church (1936) montre les premiers résultats d'indécidabilité (= non-calculabilité par une fonction récursive générale) :

- Savoir si un terme M a une forme normale est indécidable.
- Corollaire : le problème de la décision d'Hilbert (Entscheidungsproblem) n'a pas de solution.

Kleene (1935) et Turing (1937) démontrent l'équivalence entre

- les fonctions récursives générales (Herbrand, Gödel, Kleene)
- les fonctions calculables par une machine de Turing
- les fonctions définissables par un λ-terme via le codage de Church (1933) des entiers :

$$n \equiv \lambda f.\, \underbrace{f \circ \cdots \circ f}_{n \text{ fois}} \equiv \lambda f.\, \lambda x.\, \underbrace{f\ (f\ (\cdots\ (f\ x)))}_{n \text{ fois}}$$

Lambda-calcul et langages fonctionnels

Le lambda-calcul pur comme grand ancêtre et comme ≪noyau≫ des langages fonctionnels de programmation :

langage fonctionnel	=	lambda-calcul pur
	+	stratégie de réduction
	+	types de données
	+	système de types (le cas échéant)

LISP 1.5 (1960) mentionne Church et note `(LAMBDA ...)` les fonctions anonymes. Mais sa sémantique (portée dynamique des liaisons) ne respecte pas la β-réduction.

Landin (1965) propose ISWIM, un lambda-calcul étendu, comme *The next 700 programming languages*.

Dès 1970, tous les langages fonctionnels ont un noyau de lambda-calcul : Common Lisp, Scheme, SML, Caml, Haskell, …

II

La logique intuitionniste

Intuitionnisme et logique intuitionniste

Intuitionnisme : philosophie des mathématiques élaborée par L. E. J. Brouwer (1881–1966). Tous les objets mathématiques doivent être accessibles à l'intuition. Rejette les démonstrations non constructives, l'infini actuel, et l'approche axiomatique d'Hilbert.

Logique intuitionniste : famille de logiques mathématiques étudiée par Heyting, Glivenko, Gödel, Kolmogorov. Formalise l'aspect ≪démonstrations constructives uniquement≫ de l'intuitionnisme.

Démonstration constructive ou non

Comment démontrer ≪il existe $x \in A$ tel que $P(x)$≫ ?

1. De manière constructive : on définit un élement a de A à partir des hypothèses, puis on démontre la propriété $P(a)$.
2. Par l'absurde : on suppose que $\exists x \in A.\ P(x)$ est faux, c.à.d. que $\forall x \in A.\ \neg P(x)$ est vrai, et on en déduit une contradiction.

Pour un intuitionniste, la première démonstration est la seule qui prouve que $\exists x \in A.\ P(x)$ est vrai.

La seconde démonstration prouve seulement que $\exists x \in A.\ P(x)$ n'est pas faux, ce qui est un résultat plus faible.

Une aventure intuitionniste

(inspirée par G. Dowek, *Les métamorphoses du calcul*)

Dans le train de retour des Pays-Bas, on me passe un mot :

> *Les logiciens classiques français veulent votre peau. Descendez au dernier arrêt du train avant la frontière française.*
> *N.B. : le train marquera peut-être des arrêts non prévus.*

À quel arrêt descendre ?

L'ensemble A des arrêts du train hors France est 1- non vide (Bruxelles $\in A$), 2- totalement ordonné, et 3- fini, donc il contient un élément maximal : c'est ≪mon≫ arrêt.

Mais cette démonstration n'est pas constructive et ne me donne pas de critère effectif pour décider à chaque arrêt s'il faut descendre…

Tiers exclu

Règle du tiers exclu (*tertium non datur*, *excluded middle*) :

$$P \vee \neg P \quad \text{pour toute proposition } P$$

Équivalente à la règle d'élimination de la double négation :

$$\neg\neg P \Rightarrow P \quad \text{pour toute proposition } P$$

C'est le principe du raisonnement par l'absurde.

La logique intuitionniste rejette la règle du tiers exclu
(et les nombreuses règles équivalentes).

Véracité vs prouvabilité

Logique classique : toute proposition est *a priori* vrai ou fausse.
La démonstration établit lequel de ces deux cas s'applique.

D'où par exemple les raisonnements par tables de vérité pour le fragment propositionnel :

P	Q	$P \Rightarrow Q$	$Q \Rightarrow P$	$(P \Rightarrow Q) \vee (Q \Rightarrow P)$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	1	1	1

Véracité vs prouvabilité

Logique classique : toute proposition est *a priori* vrai ou fausse.
La démonstration établit lequel de ces deux cas s'applique.

Logique intuitionniste : une proposition peut être prouvée ;
ou bien sa négation peut être prouvée ;
et sinon on ne sait rien sur la proposition.

L'interprétation BHK

(Brouwer, Heyting, Kolmogorov)

La prouvabilité se caractérise par l'existence d'une construction de la proposition.

- Il existe une construction de $\top$ (le «vrai»).
- Il n'existe pas de construction de $\bot$ (le «faux»).
- Une construction de $P_1 \wedge P_2$ est un couple (c_1, c_2) où c_1 est une construction de P_1 et c_2 une construction de P_2.
- Une construction de $P_1 \vee P_2$ est un couple (i, c) où $i \in \{1, 2\}$ et c est une construction de P_i.

L'interprétation BHK

- Une construction de $P_1 \Rightarrow P_2$ est un procédé qui à partir d'une construction de P_1 produit une construction de P_2.

On peut préciser le mot ≪procédé≫ de plusieurs manières :

fonction mathématique arbitraire	(pas toujours constructif)
algorithme	(idée très intuitionniste !)
fonction récursive	($\rightarrow$ réalisabilité de Kleene)
lambda-terme	($\rightarrow$ Curry-Howard).

L'interprétation BHK

- Une construction de $\neg P \equiv P \Rightarrow \bot$ est un algorithme qui à partir d'une (hypothétique) construction de P produit un objet qui n'existe pas.

NB : l'existence d'une construction de $\neg P$ est bien plus forte que la non-existence d'une construction de P.

- Une construction de $\forall x, P(x)$ est un algorithme qui pour toute valeur de x produit une construction de $P(x)$.
- Une construction de $\exists x, P(x)$ est un couple (a, construction de $P(a)$).

Retour sur le tiers exclu

Soit TM l'ensemble des machines de Turing. On considère

$$Q \stackrel{def}{=} \forall m \in TM, \ \ \text{termine}(m) \vee \neg\text{termine}(m)$$

Q découle immédiatement du tiers exclu ($P \vee \neg P$ pour tout P).

Or, une construction de Q serait un algorithme qui, étant donnée une machine de Turing m, renvoie $(1, c)$ si m termine et $(2, c)$ sinon.
Le problème de l'arrêt étant indécidable, cet algorithme n'existe pas

Le tiers exclu n'est donc pas constructif au sens où il n'existe pas de construction qui le valide.
(Cette construction serait une procédure de décision universelle !)

Une formalisation de la logique intuitionniste

Considère des séquents intuitionnistes de la forme $\Gamma \vdash P$
(lire : ≪sous les hypothèses Γ on sait déduire P≫.)

Se compose d'axiomes (≪ce séquent est vrai≫) et de règles d'inférence (≪si les séquents du haut sont vrais alors le séquent du bas est vrai≫).
Par exemple :

$$P \vdash P$$

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}$$

Règles de la logique intuitionniste

Implication : (≪I≫ pour Introduction, ≪E≫ pour Élimination)

$$\Gamma_1, P, \Gamma_2 \vdash P \ \text{(Ax)}$$

$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q} \ (\Rightarrow \text{I}) \qquad \frac{\Gamma \vdash P \Rightarrow Q \quad \Gamma \vdash P}{\Gamma \vdash Q} \ (\Rightarrow \text{E, } \textit{modus ponens})$$

Conjonction :

$$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q} \ (\wedge \text{I}) \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P} \ (\wedge \text{E}_1) \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q} \ (\wedge \text{E}_2)$$

Règles de la logique intuitionniste

Disjonction :

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q}\ (\vee \mathrm{I}_1) \qquad \frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q}\ (\vee \mathrm{I}_2)$$

$$\frac{\Gamma \vdash P \vee Q \qquad \Gamma, P \vdash R \qquad \Gamma, Q \vdash R}{\Gamma \vdash R}\ (\vee \mathrm{E})$$

Négation :

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash P}\ (\bot \mathrm{E}, \textit{ex falso quodlibet})$$

III

Les types

Les types

La sagesse populaire :

> *On n'additionne pas des choux et des carottes.*
> *On ne mélange pas les torchons et les serviettes.*
> *Don't compare apples and oranges.*

La sagesse du physicien : les équations aux dimensions.

$d = v.t$	dist = dist	homogène, possiblement correct
$d = v/t$	dist $\neq$ dist.temps^{-2}	non homogène, forcément faux

Dans les langages de programmation

Pour détecter et rejeter des erreurs de programmation.

```
"toto"(12)              (chaîne de caractères ≠ fonction)
cos(45, "degrés")       (mauvais nombre d'arguments)
```

Comme spécifications partielles des structures de données et des interfaces de modules.

```
type 'a tree = Leaf of 'a | Node of 'a tree * 'a tree
val assoc: 'a -> ('a * 'b) list -> 'b
```

Pour clarifier le sens des programmes et faciliter la compilation.

```
integer n
real x
n = n * 2 + 1           (calcul entier)
x = x * 2 + 1           (calcul en virgule flottante)
```

Les types en logique mathématique

En mathématiques classiques : une notion intuitive.

(P.ex. en géométrie euclidienne, un point n'est pas une droite, et on ne parle jamais de l'intersection de deux points.)

En théorie des ensembles naïve : pas de types *a priori*.

P.ex. le codage des entiers $n + 1 \equiv n \cup \{n\}$.

Mais des paradoxes comme celui de Russell : $\{x \mid x \notin x\}$.

Les théories des types : mettre des types dans une logique pour empêcher les paradoxes.

> *Le typage est une espèce de surmoi interdisant certaines formes d'inceste logique du style $x \in x$.*
>
> *(J.-Y. Girard, Le point aveugle)*

La théorie des types de Russell

(1908 ; utilisée dans *Principia Mathematica*, 1910–1912)

Russell énonce un ≪principe du cercle vicieux≫ :

> *Whatever involves all of a collection must not be one of the collection.*

Pour cela, il associe un ordre (un entier) à chaque objet de la théorie, de sorte que

- ordre d'un ensemble $>$ ordre de ses éléments
  (pas de $x \in x$)
- ordre d'un quantificateur $\forall x.P >$ ordre de la variable x
  (prédicativité)

Les types ramifiés de Russell

Des types de fonctions t annotés par des entiers a :

$$
\begin{array}{lll}
t^a ::= 0^0 & & \text{proposition de base} \\
\quad \mid (t_1^{a_1}, \ldots, t_n^{a_n})^{\,a} & & \text{prédicat à } n \text{ paramètres} \\
& & \text{avec } a > \max(a_1, \ldots, a_n)
\end{array}
$$

Des règles de typage informelles mais compliquées.

Typage trop intentionnel : des formules logiques équivalentes ont des ordres différents.

Un axiome de réductibilité (pour toute formule F il existe une formule $G \Leftrightarrow F$ dont l'ordre est minimal) qui ne convainc pas.

Un effort de dé-ramification (Ramsey (1926), Hilbert et Ackermann, …) pour supprimer les ordres entiers et ne garder que les types simples (types de fonctions).

Kamareddine, Laan, Nederpelt, *A modern perspective on type theory*, partie 1.

La théorie des types simples de Church

A. Church, A Formulation of the Simple Theory of Types, *J. Symbolic Logic* 5(2), 1940.

Une grammaire des termes bien typés, indicés par leur type.

Types :	$\alpha, \beta ::=$	ι	entiers
		$\mid o$	propositions logiques
		$\mid \alpha \to \beta$	fonctions de α dans β
Termes :	$M, N ::=$	x_α	variable
		$\mid (\lambda x_\alpha .\, M_\beta)_{\alpha\to\beta}$	abstraction
		$\mid (M_{\alpha\to\beta}\ N_\alpha)_\beta$	application
		$\mid \neg_{o\to o}$	négation logique
		$\mid \wedge_{o\to o\to o}$	conjonction
		$\mid \forall_{(\alpha\to o)\to o}$	quantification universelle
		$\mid \iota_{(\alpha\to o)\to\alpha}$	un x tel que $P(x)$

Plus : α-conversion, β-conversion.

Plus : axiomatisation de l'arithmétique de Peano.

Le lambda-calcul simplement typé

La présentation moderne sépare la syntaxe des termes et le jugement de typage $\Gamma \vdash M : \alpha$
(lire : le terme M a le type α sous les hypothèses Γ).

Termes : $M, N ::= x \mid \lambda x{:}\alpha.\, M \mid M\ N$

Règles de typage :

$$\Gamma_1, x : \alpha, \Gamma_2 \vdash x : \alpha$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x{:}\alpha.\, M : \alpha \to \beta} \qquad \frac{\Gamma \vdash M : \alpha \to \beta \qquad \Gamma \vdash N : \alpha}{\Gamma \vdash M\ N : \beta}$$

Types pour les constantes utilisées par Church :

$$\neg : o \to o \qquad \forall_\alpha : (\alpha \to o) \to o$$
$$\wedge : o \to o \to o \qquad \iota_\alpha : (\alpha \to o) \to \alpha$$

Typage et réductions

Théorème (Préservation du typage par réduction)

Si $\Gamma \vdash M : \alpha$ *et* $M \to_\beta M'$ *alors* $\Gamma \vdash M' : \alpha$

Théorème (Normalisation forte)

Si $\Gamma \vdash M : \alpha$, *toute séquence de* β*-réductions issue de* M *est finie.*

En corollaire, le lambda-calcul simplement typé n'est pas Turing-complet.

IV

La correspondance de Curry-Howard

La logique combinatoire

Dès 1930, Curry étudie la logique combinatoire : une variante du λ-calcul sans l'abstraction $\lambda x.M$, mais avec des combinateurs prédéfinis par leurs règles de conversion. Par exemple :

Termes : $M, N ::= M\ N \mid S \mid K \mid I$

Règles :
$$\begin{aligned} I\ x &= x \\ K\ x\ y &= x \\ S\ x\ y\ z &= x\ z\ (y\ z) \end{aligned}$$

Tout λ-terme peut s'encoder en logique combinatoire avec une base de combinateurs adéquate, p.ex. S, K, I.

Typage de la logique combinatoire avec des types simples :

$$\frac{M : \alpha \to \beta \qquad N : \alpha}{M\ N : \beta} \qquad I : \alpha \to \alpha \qquad K : \alpha \to \beta \to \alpha$$

$$S : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma)$$

La correspondance de Curry

Pages 313–314 du livre *Combinatory logic* (1958), Curry et Feys observent que, si on lit la flèche des types $\tau_1 \to \tau_2$ comme l'implication $P \Rightarrow Q$ des propositions,

- Les types des combinateurs sont les axiomes qui définissent l'implication dans une logique à la Hilbert :

$$\begin{array}{rcll} I & : & \alpha \to \alpha & P \Rightarrow P \\ K & : & \alpha \to \beta \to \alpha & P \Rightarrow Q \Rightarrow P \\ S & : & (\alpha \to \beta \to \gamma) \to & (P \Rightarrow Q \Rightarrow R) \Rightarrow \\ & & \qquad (\alpha \to \beta) \to (\alpha \to \gamma) & \qquad (P \Rightarrow Q) \Rightarrow (P \Rightarrow R) \end{array}$$

- La règle de typage de l'application est le *modus ponens* :

$$\frac{M : \alpha \to \beta \qquad N : \alpha}{M\ N : \beta} \qquad\qquad \frac{P \Rightarrow Q \qquad P}{Q}$$

La correspondance de Curry

Via la correspondance de Curry, une proposition P est démontrable si et seulement si le type α correspondant est habité, c.à.d. qu'il existe un terme qui a le type α.

Le manuscrit de Howard

W. A. Howard, *The formulae-as-types notion of construction*, 1969

Étend et modernise le résultat de Curry :

- Lambda-calcul au lieu de logique combinatoire.
- Séquents intuitionnistes au lieu des axiomes façon Hilbert.
- Traite aussi les connecteurs $\wedge$, $\vee$, $\neg$ en plus de $\Rightarrow$; discute les quantificateurs $\forall$ et $\exists$.
- Correspondance entre la réduction des termes et l'élimination des coupures dans les démonstrations.

A circulé sous forme de photocopies de notes manuscrites.
Finalement publié (à l'identique) en 1980 dans l'ouvrage collectif *To H.B. Curry : Essays on Combinatory Logic, Lambda Calculus, and Formalism*.

Curry-Howard : l'implication

$$\Gamma_1, x : A, \Gamma_2 \vdash x : A$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\, M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

Curry-Howard : l'implication

$$\overline{\Gamma}_1, \quad A, \overline{\Gamma}_2 \vdash \quad A$$

$$\frac{\overline{\Gamma}, \quad A \vdash \quad B}{\overline{\Gamma} \vdash \quad A \Rightarrow B} \qquad \frac{\overline{\Gamma} \vdash \quad A \Rightarrow B \quad \overline{\Gamma} \vdash \quad A}{\overline{\Gamma} \vdash \quad B}$$

$\overline{\Gamma}$ est Γ sans les noms de variables, p.ex. $\overline{x : A, y : A} = A, A.$

Curry-Howard : la conjonction

On ajoute au lambda-calcul des couples et leurs projections :

Types : $A, B ::= \ldots \mid A \times B$

Termes : $M, N ::= \ldots \mid \langle M, N \rangle \mid \pi_1\, M \mid \pi_2\, M$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1\, M : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2\, M : B}$$

Curry-Howard : la conjonction

On ajoute au lambda-calcul des couples et leurs projections :

Types : $A, B ::= \ldots \mid A \times B$

Termes : $M, N ::= \ldots \mid \langle M, N \rangle \mid \pi_1\, M \mid \pi_2\, M$

$$\frac{\overline{\Gamma} \vdash A \qquad \overline{\Gamma} \vdash B}{\overline{\Gamma} \vdash A \wedge B} \qquad \frac{\overline{\Gamma} \vdash A \wedge B}{\overline{\Gamma} \vdash A} \qquad \frac{\overline{\Gamma} \vdash A \wedge B}{\overline{\Gamma} \vdash B}$$

Curry-Howard : la disjonction

On ajoute au lambda-calcul un type «somme» et les opérations correspondantes (voir cours 2e semaine) :

Types : $A, B ::= \ldots \mid A + B$

Termes : $M, N ::= \ldots \mid \mathtt{inj}_1\, M \mid \mathtt{inj}_2\, M$
$\mid \mathtt{match}\, M\, \mathtt{with}\, \mathtt{inj}_1\, x_1 \Rightarrow N_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow N_2\, \mathtt{end}$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathtt{inj}_1\, M : A + B} \qquad \frac{\Gamma \vdash N : B}{\Gamma \vdash \mathtt{inj}_2\, N : A + B}$$

$$\frac{\Gamma \vdash M : A + B \qquad \Gamma, x_1 : A \vdash N_1 : C \qquad \Gamma, x_2 : B \vdash N_2 : C}{\Gamma \vdash \mathtt{match}\, M\, \mathtt{with}\, \mathtt{inj}_1\, x_1 \Rightarrow N_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow N_2\, \mathtt{end} : C}$$

Curry-Howard : la disjonction

On ajoute au lambda-calcul un type «somme» et les opérations correspondantes (voir cours 2e semaine) :

Types : $A, B ::= \ldots \mid A + B$

Termes : $M, N ::= \ldots \mid \mathtt{inj}_1\, M \mid \mathtt{inj}_2\, M$
$\mid \mathtt{match}\, M\, \mathtt{with}\, \mathtt{inj}_1\, x_1 \Rightarrow N_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow N_2\, \mathtt{end}$

$$\frac{\overline{\Gamma} \vdash A}{\overline{\Gamma} \vdash A \vee B} \qquad \frac{\overline{\Gamma} \vdash B}{\overline{\Gamma} \vdash A \vee B}$$

$$\frac{\overline{\Gamma} \vdash A \vee B \quad \overline{\Gamma}, A \vdash C \quad \overline{\Gamma}, B \vdash C}{\overline{\Gamma} \vdash C}$$

Curry-Howard : le faux

On ajoute au lambda-calcul un type vide et un «éliminateur» pour ce type (voir cours 2e semaine) :

Types : $A, B ::= \ldots \mid \bot$

Termes : $M, N ::= \ldots \mid \texttt{match}\ M\ \texttt{with end}$

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \texttt{match}\ M\ \texttt{with end} : A}$$

(La syntaxe $\texttt{match}\ M\ \texttt{with end}$ est celle utilisée par Coq et correspond au fait que le type $\bot$ peut être défini comme un type inductif à zéro constructeurs. Plus de détails la semaine prochaine.)

Curry-Howard : le faux

On ajoute au lambda-calcul un type vide et un ≪éliminateur≫ pour ce type (voir cours 2e semaine) :

Types : $A, B ::= \ldots \mid \bot$

Termes : $M, N ::= \ldots \mid \texttt{match}\ M\ \texttt{with end}$

$$\frac{\overline{\Gamma} \vdash \quad \bot}{\overline{\Gamma} \vdash \quad A}$$

(La syntaxe $\texttt{match}\ M\ \texttt{with end}$ est celle utilisée par Coq et correspond au fait que le type $\bot$ peut être défini comme un type inductif à zéro constructeurs. Plus de détails la semaine prochaine.)

À quoi correspond la β-réduction ?

Howard (1980) observe que la β-réduction correspond à l'élimination d'une coupure dans le fragment implicatif ($\Rightarrow$) de la logique.

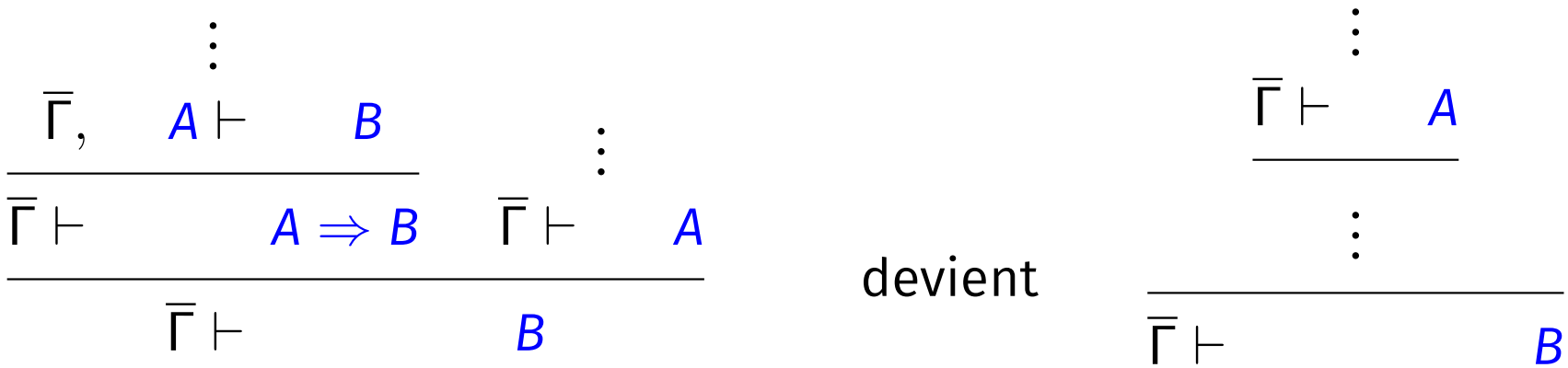
$$\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash M : B}}{\Gamma \vdash \lambda x.\, M : A \to B} \qquad \dfrac{\vdots}{\Gamma \vdash N : A}}{\Gamma \vdash (\lambda x.M)\ N : B}$$

devient

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash N : A}}{\Gamma \vdash M\{x \leftarrow N\} : B}$$

À quoi correspond la β-réduction ?

Howard (1980) observe que la β-réduction correspond à l'élimination d'une coupure dans le fragment implicatif ($\Rightarrow$) de la logique.



Les coupures

Une *coupure* est un énoncé intermédiaire (un lemme) que l'on démontre alors même qu'il n'est pas sous-formule de l'énoncé final (le théorème).

Exemple (Démonstration avec coupure)

On montre P, puis $P \Rightarrow Q$, et on conclut Q.
P est une coupure.

Exemple (Démonstration sans coupures)

On montre P, puis Q, et on conclut $P \wedge Q$.
P et Q sont sous-formules de $P \wedge Q$ et donc pas des coupures.

L'élimination des coupures

Éliminer les coupures, c'est réécrire une démonstration jusqu'à ce qu'elle ne contienne plus de coupures.

Une démonstration sans coupure est sans intérêt mathématique.
(Une belle démonstration a des lemmes et des résultats intermédiaires !)

L'élimination des coupures aide à démontrer qu'une logique est cohérente (il n'y a pas de démonstration de ≪faux≫ sans coupures).

L'élimination des coupures

Pour le fragment implicatif ($\Rightarrow$) il suffit de transformer la dérivation comme suit :

$$\dfrac{\dfrac{\vdots}{\Gamma, P \vdash Q}}{\Gamma \vdash P \Rightarrow Q}\,(\Rightarrow\mathrm{I}) \qquad \dfrac{\vdots}{\Gamma \vdash P}}{\Gamma \vdash Q}\,(\Rightarrow\mathrm{E})$$

devient

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash P}}{\vdots}}{\Gamma \vdash Q}$$

Intuition : on a un lemme P utilisé pour prouver un théorème Q. On se débarrasse du lemme P en remplaçant chaque utilisation de P dans la démonstration de Q par une copie de la démonstration de P.

Quand on ne peut plus appliquer cette transformation, la dérivation est sans coupures et a la propriété de la sous-formule.

Réductions et élimination des coupures

Howard observe qu'un terme en forme β-normale correspond à une démonstration sans coupure dans le fragment $(\Rightarrow)$ de la logique.

Le résultat s'étend aux conjonctions.

Règles de réduction pour les produits :

$$\pi_1 \langle M, N \rangle \to M \qquad \pi_2 \langle M, N \rangle \to N$$

Un terme en forme normale pour ces règles et pour β correspond à une démonstration sans coupure dans le fragment $(\Rightarrow, \wedge)$ de la logique.

Réductions et élimination des coupures

Si on ajoute $\vee$ et $\bot$, le résultat n'est plus tout à fait vrai : les règles usuelles de réduction pour les types sommes

$$\texttt{match}\ \texttt{inj}_k\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow N_1 \mid \texttt{inj}_2\ x_2 \Rightarrow N_2\ \texttt{end} \to N_k\{x_k \leftarrow M\}$$

ne suffisent pas à éliminer toutes les coupures. Il faut ajouter de nombreuses règles de réduction inhabituelles pour gérer les ≪coupures commutatives≫ (Prawitz, 1965). Ainsi, pour l'absurdité $\bot$:

$$\begin{aligned}
(\texttt{match}\ M\ \texttt{with}\ \texttt{end})\ N &\to \texttt{match}\ M\ \texttt{with}\ \texttt{end} \\
\pi_i\ (\texttt{match}\ M\ \texttt{with}\ \texttt{end}) &\to \texttt{match}\ M\ \texttt{with}\ \texttt{end} \\
\texttt{match}\ (\texttt{match}\ M\ \texttt{with}\ \texttt{end})\ \texttt{with}\ \ldots\ \texttt{end} &\to \texttt{match}\ M\ \texttt{with}\ \texttt{end}
\end{aligned}$$

Réductions et élimination des coupures

Et pour la disjonction :

$$(\texttt{match}\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow N_1 \mid \texttt{inj}_2\ x_2 \Rightarrow N_2\ \texttt{end})\ N$$
$$\to \quad \texttt{match}\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow N_1\ N \mid \texttt{inj}_2\ x_2 \Rightarrow N_2\ N\ \texttt{end}$$

$$\pi_i\ (\texttt{match}\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow N_1 \mid \texttt{inj}_2\ x_2 \Rightarrow N_2\ \texttt{end})$$
$$\to \quad \texttt{match}\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow \pi_i\ N_1 \mid \texttt{inj}_2\ x_2 \Rightarrow \pi_i\ N_2\ \texttt{end}$$

$$\begin{array}{l}\texttt{match}\ (\texttt{match}\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow N_1 \mid \texttt{inj}_2\ x_2 \Rightarrow N_2\ \texttt{end}) \\ \texttt{with}\ \texttt{inj}_1\ y_1 \Rightarrow P_1 \mid \texttt{inj}_2\ y_2 \Rightarrow P_2\ \texttt{end}\end{array}$$
$$\begin{array}{ll}\to & \texttt{match}\ M\ \texttt{with} \\ & \quad \texttt{inj}_1\ x_1 \Rightarrow (\texttt{match}\ N_1\ \texttt{with}\ \texttt{inj}_1\ y_1 \Rightarrow P_1 \mid \texttt{inj}_2\ y_2 \Rightarrow P_2\ \texttt{end}) \\ & \mid\ \texttt{inj}_2\ x_2 \Rightarrow (\texttt{match}\ N_2\ \texttt{with}\ \texttt{inj}_1\ y_1 \Rightarrow P_1 \mid \texttt{inj}_2\ y_2 \Rightarrow P_2\ \texttt{end})\end{array}$$

V

Point d’étape

Curry-Howard en 1970

Un isomorphisme entre λ-calcul simplement typé et logique intuitionniste qui met en correspondance

- types et propositions ;
- termes et démonstrations ;
- réduction et élimination des coupures.

Est-ce juste une coincidence ?

Le résultat est-il spécifique au λ-calcul simplement typé ?
Peut-il s'étendre à des langages de programmation plus riches ?

Le résultat est-il spécifique à la logique intuitionniste ?
Peut-il s'étendre à la logique classique ? à d'autres logiques ?

Des réponses dans les prochains cours !

VI

Bibliographie

Bibliographie

Familiarisation avec les idées :

- Gilles Dowek. *Les métamorphoses du calcul. Une étonnante histoire des mathématiques.* Editions Le Pommier, 2007.
- Pierre Lescanne. Et si on commençait par les fonctions ! *Images des mathématiques*, 2009.
  `http://images.math.cnrs.fr/et-si-on-commencait-par-les.html?lang=fr`

Approfondissement :

- Jean-Yves Girard. *Le point aveugle. Cours de logique. Tome 1 : vers la perfection.* Hermann, 2006. Surtout les chapitres 4 et 5.
- Benjamin Pierce. *Types and Programming Languages.* MIT Press, 2002. Surtout les chapitres 5, 9, 11 et 12.
- Morten Heine Sørensen, Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism.* 1998. Chapitres 1 à 5.
  `https://disi.unitn.it/~bernardi/RSISE11/Papers/curry-howard.pdf`.